Турсунов Баходурхон

**1. Можете ли вы объяснить разницу между процессом и потоком в контексте операционных систем?**

Процесс:

* Самостоятельный экземпляр программы с собственными ресурсами(память, файлы).
* Изоляция обеспечивает защиту и стабильность, но усложняет взаимодействие.
* «Тяжеловесный», создание и завершение требуют значительных ресурсов

Поток:

* Легковесная единица выполнения внутри процесса, разделяющая его ресурсы.
* Позволяет параллелизм и повышает производительность на многоядерных системах.
* Требует синхронизации для предотвращения конфликтов при доступе к общим данным.

Выбор:

* Процессы: для изолированных задач и приложений.
* Потоки: для повышения производительности и параллелизма в рамках одного приложения.

**2. Какие делегаты могут быть использованы при создании нового потока в .NET и что делает метод Start?**

* ThreadStart: для методов без параметров и возвращаемого значения.
* ParametrizedThreadStart: для методов с одним параметром типа object.

Метод Start() запускает выполнение потока, вызывая переданный делегат.

ThreadStart и ParametrizedThreadStart указывают, что делать потоку, а Start запускает его.

**3. Опишите общую реализацию кооперативной отмены в .NET, включая такие ключевые компоненты, как CancellationTokenSource и маркеры отмены.**

Ключевые компоненты:

* CancellationTokenSource создает маркеры отмены (CancellationToken) и управляет ими.
* CancellationToken сигнализирует о запросе отмены.
* Методы с поддержкой отмены принимают CancellationToken и периодически проверяют его состояние

Общая реализация:

* Создание. CancellationToken создается для управления отменой.
* Передача маркера. Маркер отмены (CancellationToken) передается методам, которые должны поддерживать отмену.
* Проверка отмены. Методы периодически проверяют свойство IsCancellationRequested у маркера.
* Запрос отмены. Вызов метода Cancel() у CancellationTokenSource сигнализирует о запросе отмены
* Обработка отмены. Методы, получив сигнал отмены, выполняют необходимые действия (очистка, завершение) и возвращаются.

**4. Как можно завершить выполнение потока в .NET, используя модель кооперативной отмены?**

1. Передача маркера отмены. Потому передается CancellationToken при создании или вызове метода.

2. Проверка состояния. Внутри потока периодически проверяется свойство IsCancellationRequested у маркера.

3. Реакция на отмену:

* Бросание OperationCanceledException. Сигнализирует об отмене и позволяет обработчикам исключений выполнить очистку.
* Возврат из метода. Позволяет плавно завершить выполнение.
* Установка флага. Позволяет завершить циклы или другие длительные операции.

**5. Почему важно, чтобы поток обрабатывал исключение ThreadAbortException при использовании метода Thread.Abort, и что должно быть сделано в предложении finally кода обработки?**

Thread.Abort() «Грубый» способ завершения потока, который бросает ThreadAbortException в целевом потоке.

Важно обрабатывать ThreadAbortException:

* Предотвращение неожиданного завершения. Исключение может быть перехвачено и обработано, чтобы выполнить необходимые действия перед завершением, например, очистку ресурсов или логирование.
* Избежание повреждений данных. Если не обработать исключение, поток может завершиться в непредсказуемом состоянии, что может привести к повреждению данных или несогласованности состояния приложения.

Finally. Блок finally всегда выполняется, даже если было брошено ThreadAbortException. Мы используем его для:

* Освобождения ресурсов. Закрытие файлов, соединений, освобождение памяти.
* Отката транзакций. Обеспечение целостности данных
* Логирование. Запись информации о завершении потока.

1. try {

2. // ...

3. } catch (ThreadAbortException) {

4. // Логирование, откат транзакций

5. } finally {

6. // Освобождение ресурсов

7. }

**Кратко:** Обработка ThreadAbortException и использование finally обеспечивают корректное завершение потока и предотвращают проблемы.

**Важно:** Thread.Abort() считается устаревшим и не рекомендуется к использованию. Предпочитайте кооперативную отмену.

Practice:

1. Создайте асинхронный метод, который будет выполняться асинхронно в фоновом потоке и возвращать результат операции.

2. Используя CancellationToken, добавьте возможность отменить выполнение асинхронного метода в определенный момент времени.

3. Обработайте исключения, которые могут возникнуть при выполнении асинхронных операций, с помощью блока try-catch.

4. Используйте класс Task<T> для создания асинхронного метода, который будет возвращать результат определенного типа.